HACETTEPE UNIVERSITY

Computer Science and Engineering Department

Name Surname : Ömer AKYOL
Identify Number : 20321456
Course : BIL341
Experiment : Exp4 – OS Development
Date Due : 28.11.2006
Advisors : Prof Dr.Ali Saatçi
: R. A. Şükrü Tikveş

E-Mail : b20321456@cs.hacettepe.edu.tr
Main Program : AKOS

1. Software Catalogue Information:

Programmer	**:**	Ömer Akyol
Programming Language	**:**	C / C++ Assembly
Programming Environment	**:**	Anjuta
Main Program Name	**:**	AKOS
Computer System	**:**	IMB compatible PC
Operating System	**:**	Ubuntu Linux
Source Code Length	**:**	~long

2. Software Using Documentation:

2.1 Software Usage:

When you run the OS, I name it as *AKOS* . It makes some start-up operations and open a terminal (console) screen and halts there.

AKOS's console has very basic abilities, but they do their job well. You should see a small explanation and a console symbol like ">>". When you see this symbol ">>" it means that console is waiting commands from you. If you don't know or aren't sure about, what to type. Try using "help" or "list" commands. "help" command will give you a small explanation about *AKOS* . "list" will list available commands in the console. Try using these two commands and you should have no question left in your mind. Because only a few commands are available now and they don't get any parameter like todays Linux consoles. So it is very simple to understand and use *AKOS's* console.

Keyboard support of *AKOS* is very basic. It supports US mapped keyboards. Pressing Turkish chars won't do anything. Also it has *CAPSLOCK* support (please try it). It won't light up the *CAPSLOCK* led, but it simply does its job. Also AKOS has an unfinished but partly working *SHIFT* support. As you know if *SHIFT* is pressed you get alternative characters of the keys. You will get many alternative characters true in *AKOS*. But there are some which are not mapped correctly, and they give wrong alternative char. This occurs because the keyboard map is not fully correct and I didn't have enough time to beta test all characters.

There is a reset support in *AKOS*. If you press *CTRL+ALT+DELETE* in game or in console it will reboot the OS without asking.

I have modified some *IPS* and *vga_interval* values of *Bocsh* to get a real like platform. If you get a flipping screen or an error similar to it please reset the values according to your platform. Or trying *AKOS* loaded in a diskette is the best.

2.2 Provided Possibilities:

I haven't added any new extra possibility to the OS.

2.3 Error Messages:

Unknown Command:

You will get this error if you enter a command which is not available. Try "help" or "list" commands if you don't know what to do.

3. Software Design Notes

3.1 Description of the Problem

3.1.1 Problem:

Setting up development environment:
This is a minor problem. But you can't start developing without this.

Setting up start up assembly routines IDT, ISR, GDT:
When kernel starts, it starts in protected mode and a working GDT. Starting with a protected mode was very helpful. But there is so much to do after this. First interrupts must be enabled. Enabling interrupts with *"sti"* directly will give *3 step fault error*.

The key is learning IDT, ISR and handling interrupts. I have told how I solved these problems in solution section.

Handling timer interrupts:
CPU gives 18.222 MHz frequency timer interrupt. This data must be handled to use it perfectly. And timer is essential for the game.

Keyboard interrupts:
PIC gives only chunk of scan codes to kernel. These scan codes must be handled correctly. Console and game uses keyboard too much.

Console:
In order to make a basic OS, at least a console support is needed.

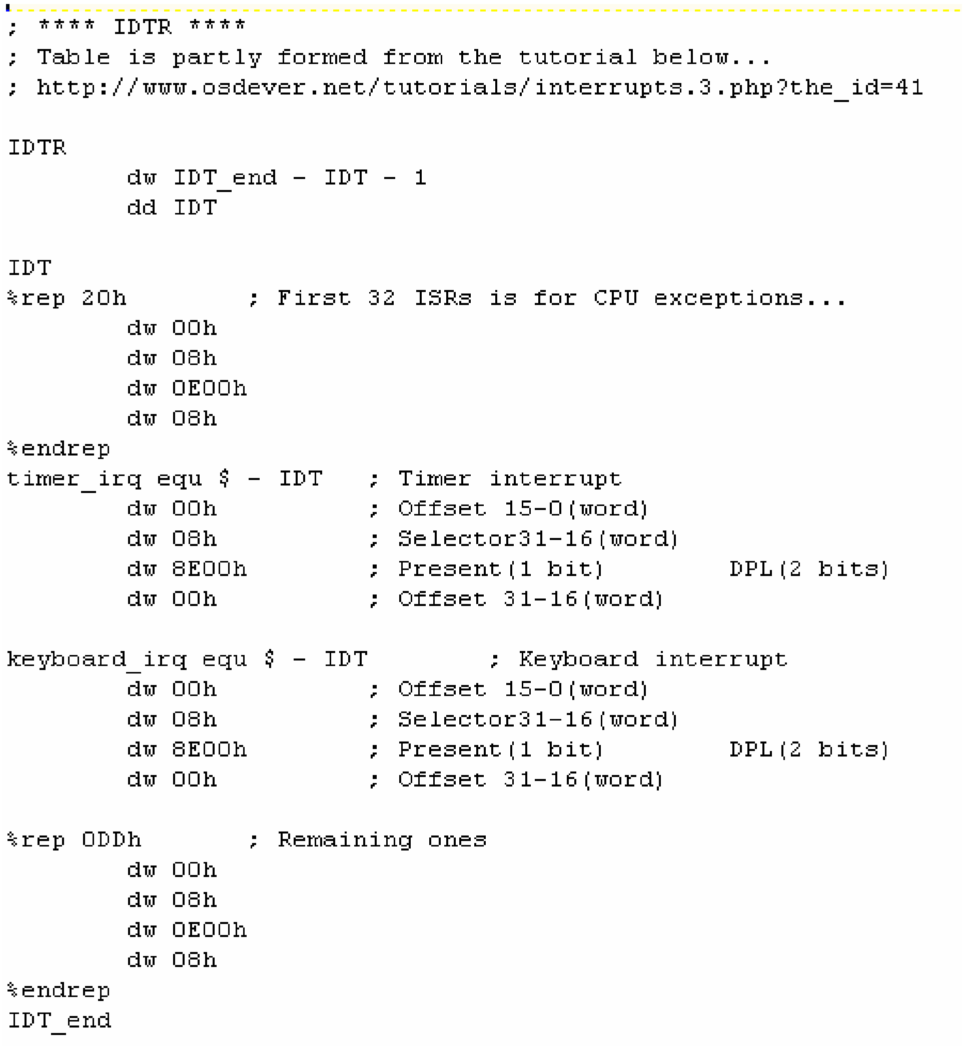
Game:
I have written too much about the design of the game in the design report. I think I have flied so high. I didn't know system's properties will be so low. I couldn't make a ground shaking game… even it is not called a game.

3.1.2 Solution:

Setting up development environment:
I am thankful to the experiment advisor for the building environment. I only typed *"make"* command and all the necessary things are set up automatically. It gave an error about *"mtools"* but I downloaded that missing package and no problems left. I used Anjuta for the development IDE. Anjuta has tab support for the source codes, and that was very helpful.

Setting up start up assembly routines IDT, ISR, GDT:

IDT is Interrupt Descriptor Table. It should be set up because, interrupts look up this table. Here is the IDT table. This table is taken from osdever.net tutorials and some kernel examples.



```
; **** IDTR ****
; Table is partly formed from the tutorial below...
; http://www.osdever.net/tutorials/interrupts.3.php?the_id=41

IDTR
        dw IDT_end - IDT - 1
        dd IDT

IDT
%rep 20h        ; First 32 ISRs is for CPU exceptions...
        dw 00h
        dw 08h
        dw 0E00h
        dw 08h
%endrep
timer_irq equ $ - IDT   ; Timer interrupt
        dw 00h          ; Offset 15-0(word)
        dw 08h          ; Selector31-16(word)
        dw 8E00h        ; Present(1 bit)        DPL(2 bits)
        dw 00h          ; Offset 31-16(word)

keyboard_irq equ $ - IDT        ; Keyboard interrupt
        dw 00h          ; Offset 15-0(word)
        dw 08h          ; Selector31-16(word)
        dw 8E00h        ; Present(1 bit)        DPL(2 bits)
        dw 00h          ; Offset 31-16(word)

%rep 0DDh       ; Remaining ones
        dw 00h
        dw 08h
        dw 0E00h
        dw 08h
%endrep
IDT_end
```

- IDT table with IRQ’s -

Two important Irq’s are installed on the IDT. Interrupts look up this table and select necessary irq.

ISR is Interrupt Service Routine. Interrupt handler routines are defined in ISR. I added only two handlers, timer and keyboard. When an timer interrupt occurs it calls *timer_handler()* from kernel (*kernel.cpp)* . And when a keyboard interrupt occurs it calls *keyboard_handler().*

Before opening all interrupts irq’s must be remapped. This is essential because Intel uses first 32 interrupt. If we don’t re map these, when time fires a timer interrupt which is 0, it will conflict with Intel’s 0. interrupt. And that is a division exception. So if we start interrupts without remapping it will directly crash.

```
/*
 Remaps IRQ's in order to prevent collusion with
 Intel's first 32 preserved CPU interrupts.

 irq0--> 21h , irq8--> 28h

 This code is taken from  Bran's Kernel Development Tutorial
 */
extern void remap_pics()
{
    outportb(0x20, 0x11);
    outportb(0xA0, 0x11);
    outportb(0x21, 0x20);
    outportb(0xA1, 0x28);
    outportb(0x21, 0x04);
    outportb(0xA1, 0x02);
    outportb(0x21, 0x01);
    outportb(0xA1, 0x01);
    outportb(0x21, 0x0);
    outportb(0xA1, 0x0);
}
```

- Remap IRQ's -

This remaps Irq's beyond 32, so there is no conflict with Intel's. This code part is taken from Bran's Kernel Development Tutorial's irq part.

Irq0 is mapped to 21h and Irq8 is mapped to 28h

Handling timer interrupts:

Irq0 is the timer interrupt. It fires 18.222Mhz frequency. I used timer in game. I made a timer class to handle time. Timer class have a tick counter inside, and in every tick it is increased. So all applications can use timer as long as they include *"timer.h"* .

For example a second delay can be made like this with this timer:

```
If(tikcs % 18 == 0) // one second passed
```

Timer is used as a count down in the game and as a small delay in moving objects…

Keyboard interrupts:

Keyboard is highly and fully used in console. Console accepts all visible characters. So this means a keyboard driver is needed. I get scan codes that keyboard interrupt produces with *keyboard_handler().* And convert these scan codes into visible Ascii chars if available.

Conversion is done in *keyboard.c* . I used a US keyboard map. I took that map from the tutorial in osdever.net. Bran's Kernel Development Tutorial's keyboard section. But I added *SHIFT* mapped chars myself.

SHIFT, ALT , CTRL and CAPSLOCK are flagged. This means when user pressed these special keys their flag is set 1, and when they released their flag is set to 0

If *SHIFT* key is 1 keyboard table with "shift map" is used, (see *keyboard.c*). And when *CAPSLOCK* is set 1 all keys are uppercased unless *SHIFT* is 0. Also when *CAPSLOCK* is set to 0 and *SHIFT* is set to 1, keys are uppers cased again. *SHIFT* works opposite with *CAPSLOCK.*

And in any time when *ALT* is 1 , *CTRL* is 1 if user presses *DELETE*, *AKOS* will reboot.

Keyboard has a small memory (512 bytes) for storing incoming chars(see *keyboard.c*). Only ASCII chars are stored without white spaces. If user presses back space last char is deleted from the buffer. Keyboard buffer is used for handling commands in console. When user presses enter keyboard buffer is read by console and handled by console.

Console:

I make a keyboard based console, this means if user doesn't press any key, console is halt. It executes when user presses a key. If the pressed key is an ASCII key, it is printed to screen.

Some simple console commands are installed into system.

help *:* Displays simple help about the OS and usage.
list *:* lists available commands
cls *:* clears screen
reboot *:* reboots *OS*
exit *:* reboots OS, it may be used for other purposes in the future.
run *:* This one is important, normally it should be run <application> but it is not. Because application support is not available yet. Only a game is available for now. It is enough to type *"run"* to start the game.

Console commands are taken from keyboard's buffer.(see *keyboard.c*) . When user presses enter, it takes buffer and handles it. And clears keyboard buffer.

Game:

Game is not COMPLETED! … I don't have enough time to complete it. We control a space ship looked thing and some enemies comes from side of the screen. They are lab courses.

We move the ship with arrow keys. And shoot with Q and W . Why there two keys for shoot? I have designed there will be a two kinds of shooting types, but no time to implement it now. But the engine is ready. See *(labwars.c and labwars.h)*

Game clears screen in every loop and makes game operations.

To quit from game press ESC. And you will return to console of *AKOS.* If you wish to play again type *"run"* in console.

Also *CTRL+ALT+DELETE* is supported in game too. Use it to reboot if you wish.

3.2 System Chart

Input	Programs	Output
<Keyboard>	<AKOS>	<Screen>

4. SOFTWARE TESTING NOTES:

4.1 Bugs and Software Reliability:

The OS is very weak but it doesn’t have known errors. Only some keyboard characters are not compatible with Turkish keyboards.

4.2 Software Extendibility and Upgradeability:

This OS can be developed as long as it can. There is no limit. Only the hard coded game parts must be cleaned, and the rest is very similar with Linus Torwald’s linux kernel. It can be improved until today’s standard Linux kernels.

And extensions can be start from the console. Now console has a few commands, it can very easily. And like Linus’s OS adding new commands are very easy.

4.3 Comments:

It was a perfect experiment; I wish I had more time… I have left many things undone, I had planned to open SVGA mode and enable buzzing sounds. Time was quite short and I was quite lazy :(.

5. REFERENCES :

- **www.osdever.net**
  - **http://www.osdever.net/bkerndev/Docs/idt.htm**
  - **http://www.osdever.net/bkerndev/Docs/isrs.htm**
  - **http://www.osdever.net/bkerndev/Docs/keyboard.htm**

- **www.osdev.org**
  - **http://www.osdev.org/osfaq2/**
  - **http://www.osdev.org/osfaq2/index.php/InterruptsForDummies**
  - **http://www.osdev.org/osfaq2/index.php/Getting%20Keyboard%20Input**